FIGURE 8 is a flowchart illustrating operations performed by the invention when handler a PMI event; and --

Please replace the paragraph beginning at page 4, line 21 with the following rewritten paragraph:

-- FIGURE 9 is a schematic diagram of a person computer system suitable for implementing the present invention. --

Please replace the paragraph beginning at page 14, line 21 with the following rewritten paragraph:

-- Next, in a decision block 108 a determination is made to whether enough SMRAM is available to hold the event handler routine. If not enough SMRAM memory space is available, the logic proceeds to a block 110 in which the caller is alerted. As an option, in response to being alerted, the caller may use the SMM_ACCESS::GetCapabilities and SMM_ACCESS::AcquireSmramRange method to acquire additional memory space within the SMRAM, as provided by a block 113. If there is not enough SMRAM memory space available, the SMRAM is closed by calling the SMM_ACCESS::Close method and an error code is returned to the caller in an error return block 114. --

Please replace the paragraph beginning at page 15, line 26 with the following rewritten paragraph:

-- With reference to FIGURE 6, the mechanism begins in a block 130, wherein the SMM_BASE driver searches all firmware volumes that are materialized in the system during pre-boot. As defined by start and end loop blocks 132 and 134, the following logic is applied to each of these firmware volumes. In a decision block 136 a determination is made to whether the firmware volume contains any firmware files conformant with the firmware file system. If the answer is no, the logic loops back to examine the next firmware volume. If one or more conformant firmware files are found, each of these files are examined using the following process, as defined by start and end loop blocks 138 and 140. In a decision block 142, the SMM_BASE drive examines

the file type of the current file to determine with it is an "SMMHandler" file. If it is not, the logic loops back to begin examination of the next file. If the file type is "SmmHandler," the SMM_BASE driver decomposes the *Sections* of the firmware file in a block 144; a section is the internal packing mechanism within a firmware file. As provided by a block 146, if a section contains a PE32+ executable image, wherein PE32+ is a Portable Executable image type described by Microsoft in the Portable Image specification (posted on the Internet at "www.microsoft.com/hwdev/efi") that is of the same machine type as which the SMM_BASE is implemented (e.g., the computer system is an IA32 machine and the handler is an IA32 PE32+ image) or if the SMM_BASE implementation is on an IA32 system that supports loading legacy 16-bit handlers, the SMM_BASE driver shall install the executable image or legacy 16-bit handler contained in the section. The logic then proceeds to process subsequent firmware files and firmware volumes in a similar manner. --

Please replace the paragraph beginning at page 18, line 5 with the following rewritten paragraph:

-- Once the machine state(s) of the processor(s) has/have been saved, native 64-bit handlers are dispatched in order until an appropriate event handler is executed to completion to service the PMI event, as provided by start loop and end loop blocks 162 and 163. As before, in one embodiment the event handlers are stored as a linked list that is traversed in order from top to bottom, wherein a first event handler is dispatched and additional event handlers are dispatched as needed. Each event handler contains a first portion of code that is used to determine if that handler is the proper handler for servicing the xMI event, as provided by a decision block 164, which may typically include interrogation of a corresponding hardware component in the manner discussed above. If a currently executed event handler is determined to be the appropriate handler, that event handler is executed to completion in a block 165, whereupon it returns a code to the SMM Nub indicating that it has serviced the PMI event in a return block 166. If the event handler determines that it is not the appropriate handler for the PMI event, it returns a code to the SMM Nub indicating such, and the SMM Nub dispatches the next event handler in the list. In a manner similar to